# 11. Visible-Surface Detection Methods

Problem definition of Visible-Surface Detection Methods:
To identify those parts of a scene that are visible from a chosen viewing position.
Surfaces which are obscured by other opaque surfaces along the line of sight (projection) are invisible to the viewer.

Characteristics of approaches:
- Require large memory size?
- Require long processing time?
- Applicable to which types of objects?

Considerations:
- Complexity of the scene
- Type of objects in the scene
- Available equipment
- Static or animated?

Classification of Visible-Surface Detection Algorithms:

1. Object-space Methods

Compare objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible:

For each object in the scene do
Begin
1. Determine those parts of the object whose view is unobstructed by other parts of it or any other object with respect to the viewing specification.
2. Draw those parts in the object color.

End

- Compare each object with all other objects to determine the visibility of the object parts.
- If there are n objects in the scene, complexity = $O(n^2)$
- Calculations are performed at the resolution in which the objects are defined (only limited by the computation hardware).
- Process is unrelated to display resolution or the individual pixel in the image and the result of the process is applicable to different display resolutions.
- Display is more accurate but computationally more expensive as compared to image space methods because step 1 is typically more complex, eg. Due to the possibility of intersection between surfaces.
- Suitable for scene with small number of objects and objects with simple relationship with each other.

2. Image-space Methods (Mostly used)

Visibility is determined point by point at each pixel position on the projection plane.

For each pixel in the image do
Begin
1. Determine the object closest to the viewer that is pierced by the projector through the pixel
2. Draw the pixel in the object colour.

End

- For each pixel, examine all n objects to determine the one closest to the viewer.
- If there are p pixels in the image, complexity depends on n and p ( O(np) ).

- Accuracy of the calculation is bounded by the display resolution.
- A change of display resolution requires re-calculation.

Application of Coherence in Visible Surface Detection Methods:

- Making use of the results calculated for one part of the scene or image for other nearby parts.
- Coherence is the result of local similarity
- As objects have continuous spatial extent, object properties vary smoothly within a small local region in the scene. Calculations can then be made incremental.

Types of coherence:

1. Object Coherence:

Visibility of an object can often be decided by examining a circumscribing solid (which may be of simple form, eg. A sphere or a polyhedron.)

2. Face Coherence:

Surface properties computed for one part of a face can be applied to adjacent parts after small incremental modification. (eg. If the face is small, we sometimes can assume if one part of the face is invisible to the viewer, the entire face is also invisible).

3. Edge Coherence:

The Visibility of an edge changes only when it crosses another edge, so if one segment of an non-intersecting edge is visible, the entire edge is also visible.

4. Scan line Coherence:

Line or surface segments visible in one scan line are also likely to be visible in adjacent scan lines. Consequently, the image of a scan line is similar to the image of adjacent scan lines.

5. Area and Span Coherence:

A group of adjacent pixels in an image is often covered by the same visible object. This coherence is based on the assumption that a small enough region of pixels will most likely lie within a single polygon. This reduces computation effort in searching for those polygons which contain a given screen area (region of pixels) as in some subdivision algorithms.

6. Depth Coherence:

The depths of adjacent parts of the same surface are similar.

7. Frame Coherence:

Pictures of the same scene at successive points in time are likely to be similar, despite small changes in objects and viewpoint, except near the edges of moving objects.

Most visible surface detection methods make use of one or more of these coherence properties of a scene.

To take advantage of regularities in a scene, eg., constant relationships often can be established between objects and surfaces in a scene.

## 11.1 Back-Face Detection

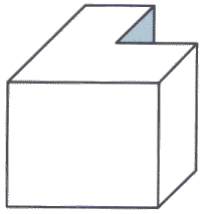
In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces which are opposite to the viewer (back faces).

These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test.

Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer.

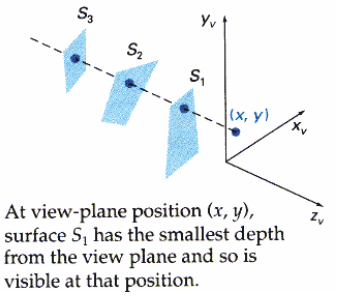
The test is very simple, suppose the z axis is pointing towards the viewer, if the z component of the normal vector is negative, then, it is a back face. If the z component of the vector is positive, it is a front face.

Note that this technique only caters well for nonoverlapping convex polyhedra.



For other cases where there are concave polyhedra or overlapping objects, we still need to apply other methods to further determine where the obscured faces are partially or completely hidden by other objects (eg., using Depth-Buffer Method or Depth-Sort Method).

## 11.2 Depth-Buffer Method (Z-Buffer Method)



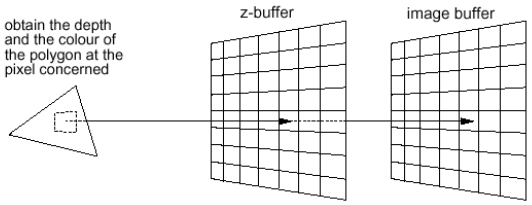

At view-plane position $(x, y)$, surface $S_1$ has the smallest depth from the view plane and so is visible at that position.

This approach compares surface depths at each pixel position on the projection plane.

Object depth is usually measured from the view plane along the z axis of a viewing system.

This method requires 2 buffers: one is the image buffer and the other is called the z-buffer (or the depth buffer). Each of these buffers has the same resolution as the image to be captured.

As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x,y) position.



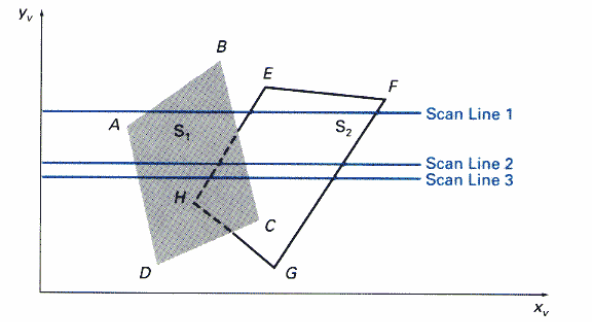

Algorithm:

1. Initially each pixel of the z-buffer is set to the maximum depth value (the depth of the back clipping plane).
2. The image buffer is set to the background color.
3. Surfaces are rendered one at a time.
4. For the first surface, the depth value of each pixel is calculated.
5. If this depth value is smaller than the corresponding depth value in the z-buffer (ie. it is closer to the view point), both the depth value in the z-buffer and the color value in the image buffer are replaced by the depth value and the color value of this surface calculated at the pixel position.
6. Repeat step 4 and 5 for the remaining surfaces.
7. After all the surfaces have been processed, each pixel of the image buffer represents the color of a visible surface at that pixel.

- This method requires an additional buffer (if compared with the Depth-Sort Method) and the overheads involved in updating the buffer. So this method is less attractive in the cases where only a few objects in the scene are to be rendered.
- Simple and does not require additional data structures.
- The z-value of a polygon can be calculated incrementally.
- No pre-sorting of polygons is needed.
- No object-object comparison is required.
- Can be applied to non-polygonal objects.
- Hardware implementations of the algorithm are available in some graphics workstation.
- For large images, the algorithm could be applied to, eg., the 4 quadrants of the image separately, so as to reduce the requirement of a large additional buffer.

### 11.3 Scan-Line Method

In this method, as each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the intensity value for that position is entered into the image buffer.

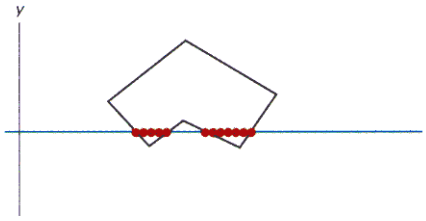



Scan lines crossing the projection of two surfaces, $S_1$ and $S_2$, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

```
For each scan line do
    Begin
        For each pixel (x,y) along the scan line do          ---------- Step 1
        Begin
            z_buffer(x) = 0
            Image_buffer(x,y) = background_color
        End

        For each polygon in the scene do                     ---------- Step 2
        Begin
            For each pixel (x,y) along the scan line that is covered by the polygon do
            Begin
                2a. Compute the depth or z of the polygon at pixel location (x,y).
                2b. If z < z_buffer(x) then
                        Set z_buffer(x) = z
                        Set Image_buffer(x,y) = polygon's colour
            End
        End
    End
```

- Step 2 is not efficient because not all polygons necessarily intersect with the scan line.
- Depth calculation in 2a is not needed if only 1 polygon in the scene is mapped onto a segment of the scan line.
- To speed up the process:

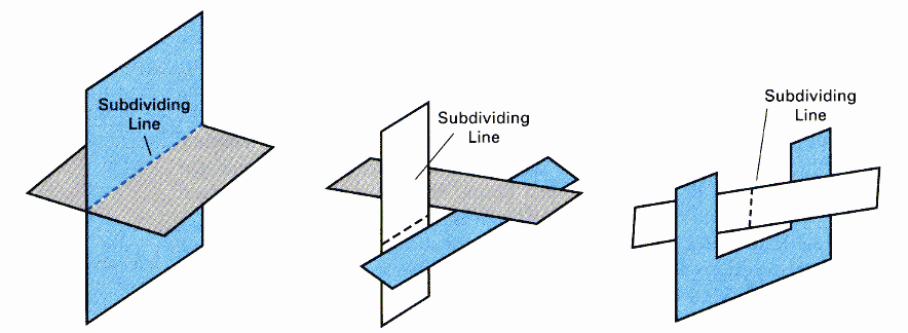
> Recall the basic idea of polygon filling: For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are sorted from left to right. Then, we fill the pixels between each intersection pair.
>
> 
> 

With similar idea, we fill every scan line span by span. When polygons overlap on a scan line, we perform depth calculations at their edges to determine which polygon should be visible at which span.

Any number of overlapping polygon surfaces can be processed with this method. Depth calculations are performed only when there are polygons overlapping.

We can take advantage of coherence along the scan lines as we pass from one scan line to the next. If there is no change in the pattern of the intersection of polygon edges with the successive scan lines, it is not necessary to do depth calculations.

This works only if surfaces do not cut through or otherwise cyclically overlap each other. If cyclic overlap happens, we can divide the surfaces to eliminate the overlaps.



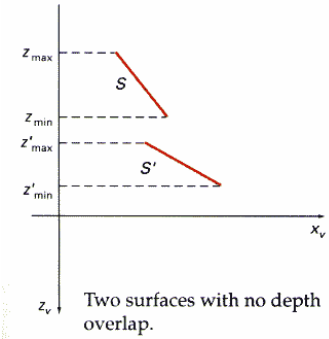

- The algorithm is applicable to non-polygonal surfaces.
- Memory requirement is less than that for depth-buffer method.
- Lot of sortings are done on x-y coordinates and on depths.

## 11.4 Depth-Sort Method

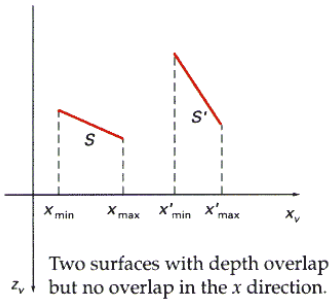
1. Sort all surfaces according to their distances from the view point.
2. Render the surfaces to the image buffer one at a time starting from the farthest surface.
3. Surfaces close to the view point will replace those which are far away.
4. After all surfaces have been processed, the image buffer stores the final image.

The basic idea of this method is simple. When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming.

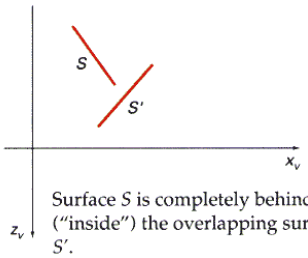
Example: Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.



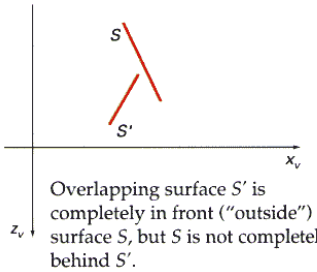
Two surfaces with no depth overlap.



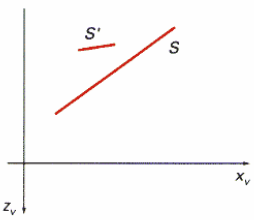
Two surfaces with depth overlap but no overlap in the $x$ direction.



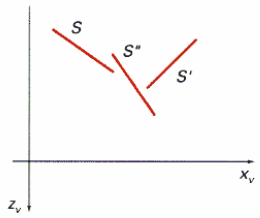
Surface $S$ is completely behind ("inside") the overlapping surface $S'$.



Overlapping surface $S'$ is completely in front ("outside") of surface $S$, but $S$ is not completely behind $S'$.



Surface $S$ has greater depth but obscures surface $S'$.



Three surfaces entered into the sorted surface list in the order $S$, $S'$, $S''$ should be reordered $S'$, $S''$, $S$.

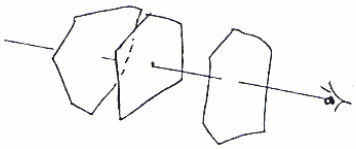
In case if there are any overlaps in depth, we need to make some additional comparisons to determine whether a pair of surfaces should be reordered. The checking is as follows:

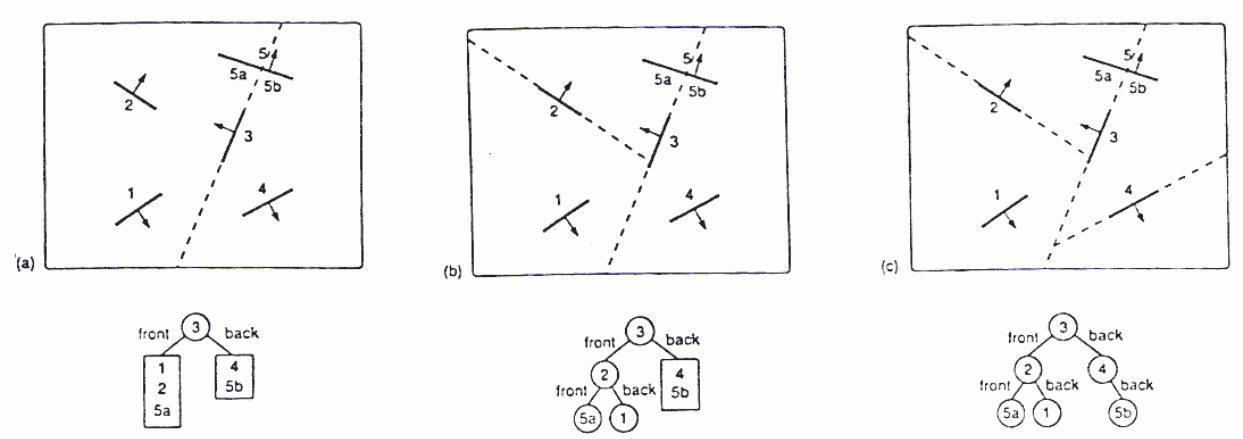
a. The bounding rectangles in the xy plane for the 2 surfaces do not overlap
b. The surface S with greater depth is completely behind the overlapping surface relative to the viewing position.
c. The overlapping surface is completely in front of the surface S with greater depth relative to the viewing position.
d. The projections of the 2 surfaces onto the view plane do not overlap.

If any of the above tests is passed, then the surfaces no need to be re-ordered.

## 11.5 Binary Space Partitioning

- suitable for a static group of 3D polygons to be viewed from a number of view points.
- based on the observation that hidden surface elimination of a polygon is guaranteed if all polygons on the other side of it as the viewer is painted first, then itself, then all polygons on the same side of it as the viewer.






1. The algorithm first builds the BSP tree:
- a root polygon is chosen (arbitrarily) which divides the region into 2 half-spaces (2 nodes => front and back)
- a polygon in the front half-space is chosen which divides the half-space into another 2 half-spaces
- the subdivision is repeated until the half-space contains a single polygon (leaf node of the tree)
- the same is done for the back space of the polygon.

2. To display a BSP tree:
- see whether the viewer is in the front or the back half-space of the root polygon.
- if front half-space then first display back child (subtree) then itself, followed by its front child / subtree
- the algorithm is applied recursively to the BSP tree.

BSP Algorithm

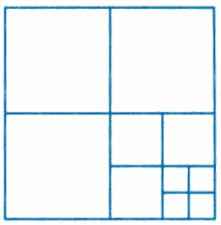
```
Procedure DisplayBSP(tree: BSP_tree)
Begin
    If tree is not empty then
        If viewer is in front of the root then
            Begin
                DisplayBSP(tree.back_child)
                displayPolygon(tree.root)
                DisplayBSP(tree.front_child)
            End
        Else
            Begin
                DisplayBSP(tree.front_child)
                displayPolygon(tree.root)
                DisplayBSP(tree.back_child)
            End
End
```

Discussion:
- Back face removal is achieved by not displaying a polygon if the viewer is located in its back half-space
- It is an object space algorithm (sorting and intersection calculations are done in object space precision)
- If the view point changes, the BSP needs only minor re-arrangement.
- A new BSP tree is built if the scene changes
- The algorithm displays polygon back to front (cf. Depth-sort)

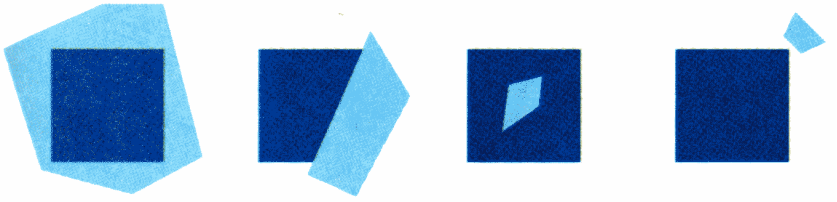
## 11.6 Area Subdivision Algorithms

The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface.

The total viewing area is successively divided into smaller and smaller rectangles until each small area is simple, ie. it is a single pixel, or is covered wholly by a part of a single visible surface or no surface at all.



Dividing a square area into equal-sized quadrants at each step.




Possible relationships between polygon surfaces and a rectangular area.

The procedure to determine whether we should subdivide an area into smaller rectangle is:

1. We first classify each of the surfaces, according to their relations with the area:

Surrounding surface - a single surface completely encloses the area
Overlapping surface - a single surface that is partly inside and partly outside the area
Inside surface - a single surface that is completely inside the area
Outside surface - a single surface that is completely outside the area.

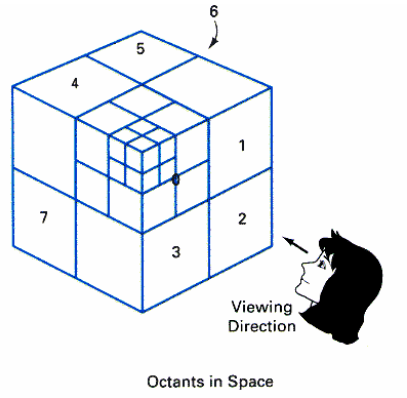
To improve the speed of classification, we can make use of the bounding rectangles of surfaces for early confirmation or rejection that the surfaces should be belong to that type.

2. Check the result from 1., that, if any of the following condition is true, then, no subdivision of this area is needed.

a. All surfaces are outside the area.
b. Only one surface is inside, overlapping or surrounding surface is in the area.
c. A surrounding surface obscures all other surfaces within the area boundaries.

For cases b and c, the color of the area can be determined from that single surface.

## 11.7 Octree Methods

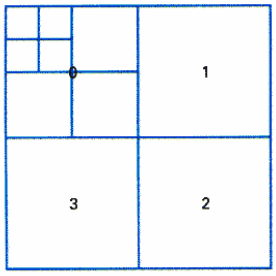


Octants in Space

In these methods, octree nodes are projected onto the viewing surface in a front-to-back order. Any surfaces toward the rear of the front octants (0,1,2,3) or in the back octants (4,5,6,7) may be hidden by the front surfaces.

With the numbering method (0,1,2,3,4,5,6,7), nodes representing octants 0,1,2,3 for the entire region are visited before the nodes representing octants 4,5,6,7. Similarly the nodes for the front four sub-octants of octant 0 are visited before the nodes for the four back sub-octants.

When a colour is encountered in an octree node, the corresponding pixel in the frame buffer is painted only if no previous color has been loaded into the same pixel position.



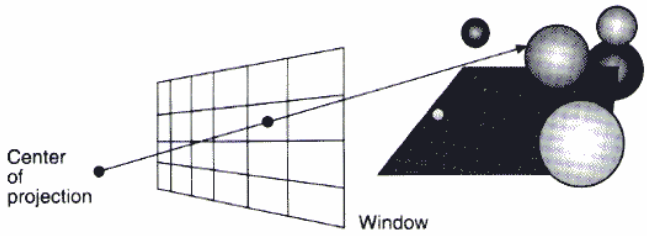
Quadrants for
the View Plane

In most cases, both a front and a back octant must be considered in determining the correct color values for a quadrant. But

- If the front octant is homogeneously filled with some color, we do not process the back octant.
- If the front is empty, it is necessary only to process the rear octant.
- If the front octant has heterogeneous regions, it has to be subdivided and the sub-octants are handled recursively.

## 11.8 Ray-Casting Method

The intensity of a pixel in an image is due to a ray of light, having been reflected from some objects in the scene, pierced through the centre of the pixel.



A ray is fired from the center of projection through each pixel to which the window maps, to determine the closest object intersected.

So, visibility of surfaces can be determined by tracing a ray of light from the centre of projection (viewer's eye) to objects in the scene. (backward-tracing).

⇨ Find out which objects the ray of light intersects.
⇨ Then, determine which one of these objects is closest to the viewer.
⇨ Then, set the pixel color to this object.

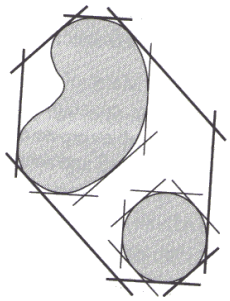
The ray-casting approach is an effective visibility-detection method for scenes with curved surfaces, particularly spheres.

Speeding up the intersection calculation in ray casting

For 1024x1024 pixels image and 100 objects in the scene,
total number of object intersection calculations is about 100 millions.

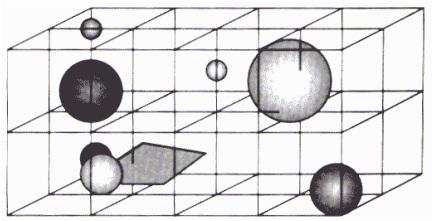
1. Bounding Volume Approach

- Test for intersection of the ray with the object's bounding volume.
- Typical bounding volumes are sphere, ellipsoid, rectangular solid. The intersection calculation for these bounding volumes is usually faster than the displayed object.
- If the ray does not intersect the bounding volume, no further processing of the object is needed.
- Other bounding volumes include convex polygons formed by a set of planes.



2. Using Hierarchies
- If a parent bounding volume does not intersect with a ray, all its children bounding volumes do not intersect with the ray and need not be processed
- Thus reduce the number of intersection calculations.

3. Space Partitioning Approach



- Partition the space into a regular grid of equal-size volumes.
- Each volume has associated with it a list of objects which are contained within or intersect the volume.
- Intersection calculation is only applied to those objects that are contained within the partition through which the ray passes.
- Objects lying within the partitions which do not intersect with the ray are not processed.

**11.9 Summary and Comparison**

The most appropriate algorithm to use depends on the scene
- depth-sort is particularly suited to the scenes with objects which are spreading out along the z-axis and/or with a small number of objects => rarely overlap in depth
- scan-line and area subdivision algorithms are suitable to the scenes where objects are spreading out horizontally and/or the scenes with small number of objects (about several thousand surfaces).
- Z-buffer and subdivision algorithms perform best for scene with fewer than a few thousand surfaces.
- Octree is particularly good because it does not require any pre-sorting or intersection calculation.
- If parallel processing hardware is available, ray casting would be a good choice (each processor handles a ray).